A Generator for Large Rdist distfiles

Rdist is a tool for UNIX systems which makes it easy to distribute collections of files between different systems. Unfortunately, when you start dealing with hundreds of systems which all need a large number of software packages, the file that controls the behavior of rdist becomes extremely large and hard to manage. My tool allows a system administrator to describe packages, groups of packages, and attribute these to various different systems in an intermediate language that makes for much shorter, easier to manage files.

Here is a short example of what the input language looks like:

```plaintext
package ( vim ) (  
    source ( vim )  
    dest ( /usr/local/vim )  
    special ( ln -s /usr/local/vim/bin/vim /usr/local/bin/vim )  
)  

package ( emacs ) (  
    source ( emacs )  
    dest ( /usr/local/emacs )  
    special (  
        ln -s /usr/local/emacs/bin/emacs /usr/local/bin/emacs  
)  
)  

package ( lynx ) (  
    source ( lynx )  
    dest ( /usr/local/lynx )  
)  

package ( motd ) (  
    source ( netfiles/motd )  
    dest ( /tmp/motd )  
    special ( cat /etc/motd.local /tmp/motd > /etc/motd )  
)  

group ( Editors ) (  
    vim  
    emacs  
)  

host ( simlab ) (  
    arch ( i386-unknown-redhatEnterprise3.0 ) (  
        Editors  
        lynx  
        motd  
)
This file would produce a much longer file, listing all the details attributed to a package for each system it is sent to. For instance,

```
vim:
  ( /distrib/i386-unknown-redhatEnterprise3.0/vim ) -> ( simlab )
  install /usr/local/vim;
  special "ln -s /usr/local/vim/bin/vim /usr/local/bin/vim ";
```

Following are the details of the specification and implementation.

The language is broken up into simple chunks of non-whitespace characters as described in Dist.gla:

```
$ # *\n [hashComment]

Chunk: $[a-zA-Z0-9,._/]{1}*]+ [mkidn]
```

Host names, architecture descriptions, package and group names, and the components of the special shell script blocks are built from these Chunks.

The primary entities of concern in this language are groups and packages. Packages themselves hold the properties of their source and destination paths, and the optional shell script to be execute upon their installation. Each group has a list attributed to it which can contain any number of packages and other groups. A package's default settings described in its package definition can be overwritten in a group context. Groups and packages share the same namespace and thus the same definition property to make sure only one package or group is attributed to one identifier. These features are described in Dist.pdl:

```
Def: int;
PSrc, PDest: int;
PSpec: lst_ptr;
List: lst_ptr;
"backend.h"
```

Psrrc and Pdest hold keys pointing to the string storage module. Lst_ptr is a pointer to a general purpose linked list structure defined in backend.h. It is used to store both lists
of strings, specifically the lists of host names in host and clients blocks as well as tokens within special blocks, and lists of keys corresponding to packages and groups in any context where there is a list of software.

The language, of course, is detailed in Dist.lido, annotated as follows:

CHAIN ready: int;
ATTR go: int;
RULE: Root ::= Document
    COMPUTE
        CHAINSTART Document.ready = 0;
        Root.go = 1 <- Document.ready;
    END;

The chain within the root node is there simply to ensure that all computations are done before anything is printed out. This was done so errors wouldn't occur halfway through printing the output in the case of an error somewhere within a host.

RULE: Document LISTOF Package | Group | Host END;

ATTR Sym: int;
ATTR arch: int;
ATTR Key: DefTableKey;

In many instances, arch is zero. However, it is meant to correspond to the tokens that hold architecture names for software being installed and will always be assigned at the very root of a list of software being installed on a system.

CHAIN hosts, pkgs: lst_ptr;
CHAIN psrc, pdest: int;
CHAIN pspec: lst_ptr;

RULE: Package ::= 'package' '(' 'PkgName' ')' '(' 'PkgAttrList '
    COMPUTE
        Package.Sym = PkgName.Sym;
        CHAINSTART PkgAttrList.psrc = 0;
        CHAINSTART PkgAttrList.pdest = 0;
        CHAINSTART PkgAttrList.pspec = NULL;
        SetDef( Package.Key, 1, 2 );
        IF( EQ( GetDef( Package.Key, 1 ), 2 ),
            errx( 1, "*** multiple definitions of %s", StringTable
                ( PkgName.Sym ) ) );
        IF( EQ( PkgAttrList.psrc, 0 ),
            errx( 1, "*** package %s has no defined source", StringTable
                ( PkgName.Sym ) ) );
        IF( EQ( PkgAttrList.pdest, 0 ),

IF( EQ( PkgAttrList.pspec, 0 ),

errx( 1, "*** package %s has no defined destination", StringTable ( PkgName.Sym ) )
);

ResetPsrc( Package.Key, PkgAttrList.psrc );
ResetPdest( Package.Key, PkgAttrList.pdest );
ResetPSpec( Package.Key, PkgAttrList.pspec );
END;

The parser dives into the list containing the details of the package, then checks to make
sure the path and destinations have been set. If either has not been set, the program
aborts.

RULE: Group ::= 'group' '(' PkgName ')' '('
      SoftwareList
')' COMPUTE
      Group.Sym = PkgName.Sym;

      CHAINSTART SoftwareList.pkgs = startlist();

      SoftwareList.arch = 0;

      SetDef( Group.Key, 1, 2 );
      IF( EQ( GetDef( Group.Key, 1 ), 2 ),
          errx( 1, "*** multiple definitions of %s", StringTable ( PkgName.Sym ) ) );

          ResetList( Group.Key, SoftwareList.pkgs );
      END;

startlist() simply returns NULL, but it denotes the end of a linked list. Computations
done under SoftwareList build these lists.

RULE: Host ::= 'host' '(' HostNameList ')' '('
      HostArchList
      ClientsList
')' COMPUTE
      CHAINSTART HostNameList.hosts = startlist();
      CHAINSTART HostArchList.pkgs = startlist();

      Host.ready = 1 <- (HostArchList.hosts, HostArchList.pkgs);

      writelist( HostNameList.hosts, HostArchList.pkgs ) <- INCLUDING
      Root.go;
      END;

Once the full list of host names and the collection of software destined to be sent to them
have been created, writelist() is called to map these two collections of data together
and actually produce the necessary output.

RULE: ClientsList LISTOF Clients END;
Here, each list of host names is built like a stack.

The way client lists are built is essentially identical to host blocks. They are created this way simply for organizational purposes within the source file.
Here lists of packages for both groups and installation within hosts and clients are built. If the package name is followed by (), it is interpreted as a list of overriding options. A new entity is created inheriting the original attributes attributed to the package, then they are each changed during the parsing of the PkgAttrList. If the package is a group, an error is reported and the program aborts.

RULE: PkgAttrList LISTOF PkgSource | PkgDest | PkgSpecial END;

RULE: PkgSource ::= 'source' '(' Path ')'"""
    COMPUTE
    PkgSource.psrc = Path.Sym;
END;
RULE: PkgDest ::= 'dest' '(' Path ')'"""
    COMPUTE
    PkgDest.pdest = Path.Sym;
END;

CHAIN stoklst: lst_ptr;
RULE: PkgSpecial ::= 'special' '(' ShellScript ')'"""
    COMPUTE
    CHAINSTART HEAD.stoklst = startlist();

    PkgSpecial.pspec = ShellScript.stoklst;
END;

RULE: ShellScript LISTOF ShellToken END;

RULE: ShellToken ::= Chunk
    COMPUTE
    ShellToken.stoklst = addtolist( ShellToken.stoklst, NoKey, Chunk, 0 );
END;
The shell script in special blocks is interpreted simply as a list of Chunks.

RULE: Path ::= Chunk
    COMPUTE
    Path.Sym = Chunk;
    END;
RULE: PkgName ::= Chunk
    COMPUTE
    PkgName.Sym = Chunk;
    END;
RULE: ArchName ::= Chunk
    COMPUTE
    ArchName.Sym = Chunk;
    END;
RULE: HostName ::= Chunk
    COMPUTE
    HostName.Sym = Chunk;
    END;

Here tokens are assigned to more useful constructs.

SYMBOL Package INHERITS IdDefScope END;
SYMBOL PkgEnt INHERITS IdUseEnv END;

SYMBOL Group INHERITS IdDefScope END;
SYMBOL GrpEnt INHERITS IdUseEnv END;

Packages and groups are the only important identifiers in this language.

Example input files, including broken ones, are included in the zip file.